第1章

HDLによるLSI設計を体験する

——2005年1月号付属FPGA基板を使って“演算当てゲーム”を作る

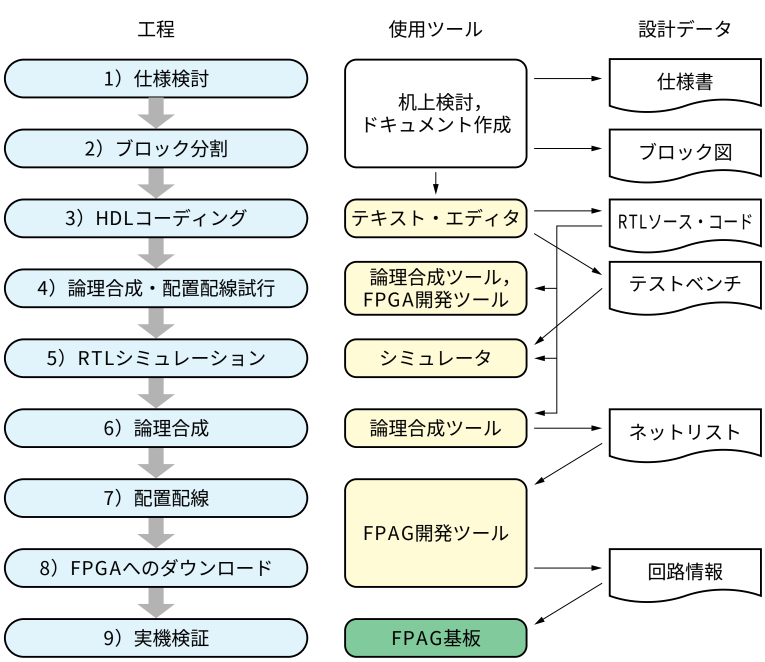
横溝憲治

ここでは，FPGA（field programmable gate array）を使って簡単なゲームを作りながら，LSI設計の手順を学ぶ．本稿の前半では，FPGA設計の基本的な流れやFPGA開発ツール，HDL設計について説明する．後半では，本誌2005年1月号に付属したFPGA基板を使って“演算当てゲーム”を実現する手順を解説する．なお，本稿で紹介するHDLソースやファイルは，本誌付属のCD-ROMに収録されている．（編集部）

現在の電子機器は，日を追うごとに複雑化しています．それに伴って，電子機器を開発する際の作業量が膨大となり，設計現場では分業化が進んできました．例えば，ディジタルLSI設計の分野では，論理回路設計者はシミュレーションで回路の動作を追うだけの日々を送っており，実際に動作しているLSIに触れる機会が少なくなっているように思います．

筆者は，このような状況は設計者にとってたいへん不幸なことと感じています．設計者は，自分で開発したハードウェアを実際に動かす体験によって，多くのノウハウを得ることができます．そして，なにより「ハードウェアを設計している」という実感を得ることが重要だと思います．幸い，FPGAやComplex PLDの普及により，オリジナルの回路を手軽に動作させることが可能となってきています．

今回は，本誌2005年1月号に付属していたFPGA基板を使って，HDLコーディングから実際に回路を動作させるまでの手順を紹介します．なお，本稿で利用するサンプルのHDLコードやファイルは，本誌付属のCD-ROMに収録されています．



◀ **図1**
FPGA開発フローと使用ツール
今回使用するFPGA開発ツール「ISE WebPACK」は，テキスト・エディタや論理合成の機能も備えている．

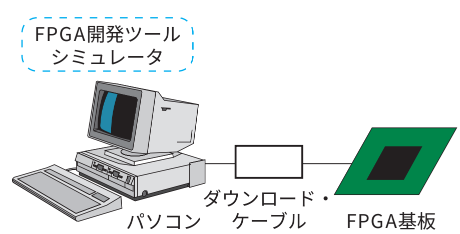


図2　FPGA開発環境
パソコン上で設計データを作成して，ダウンロード・ケーブルを介してFPGAへ回路情報（ビットストリーム）を転送する．回路情報をROMに書き込めば，パソコンなしでFPGAへデータを転送できる．

1 FPGA設計と開発ツール

ディジタル回路の設計では，HDL（hardware description language；ハードウェア記述言語）を用いる手法が主流です．C言語ベースの設計など，新しい設計手法も採用され始めていますが，ここしばらくはHDLを用いた設計・検証が主流であり続けるでしょう．

● FPGA設計フローは九つの工程から成る

図1は一般的なFPGA設計のフローです．図を追いながら，フローを簡単に見ていきましょう．

1）仕様検討

これから作成する回路の機能を決めます．実際の設計では要求仕様が提示されるので，制約事項を踏まえて設計に着手できるレベルまで仕様を詳細に決めます．

2）ブロック分割

機能別にブロック分割を行い，ブロック間の接続がわかるブロック図を描きます．そして，ブロック間のインターフェースを決定しておきます．

3）HDLコーディング

各ブロックのHDLをRTL（register transfer level；レジスタ転送レベル）のスタイルで記述します．

4）論理合成・配置配線試行

FPGA化できないHDLコードをシミュレーションしても意味がありません．まずは記述したコードに問題がないかどうかをFPGA開発ツールでチェックします．

5）RTLシミュレーション

テストベンチを作成して，シミュレーションでHDLコードの論理検証を行います．

6）論理合成

論理合成ツールを使用してRTLのHDLコードをゲート・レベルのネットリストに変換します．

7）配置配線

論理合成によって生成したネットリストに従って，FPGA開発ツールで部品を配置し，配線します．必要に応じてバック・アノテーション（配置配線後の遅延情報を反映したシミュレーション）を実施します．

8）FPGAへのダウンロード

FPGA開発ツールでFPGAへ渡す回路情報（ビットストリーム）を生成し，ダウンロード・ケーブルを介して回路情報をFPGAへ転送します．

9）実機検証

FPGAを実際に動作させて検証します．

● 無償または安価な開発環境がそろっている

次にFPGAの開発環境について説明します．図2はパソコンを利用した開発環境です．パソコン上で動作するFPGA開発ツールで設計し，FPGAへの回路情報のダウンロードもパソコンから行います．用意するものはパソコン，FPGA開発ツール，ダウンロード・ケーブル，FPGAが実装されている基板です．

開発ツールは回路規模や機能の制限はありますが，多くのFPGAベンダが無償で提供しています．無償で提供されているツールは単なるお試し用のバージョンではなく，実際の開発に必要な機能をひととおり備えています．これらのツールによってかなりの回路規模まで開発できるので，個人利用であれば困ることはありません．また，有償版ツールが必要な場合でも価格設定は低く抑えられているので，会社での導入も容易でしょう．

パソコンはWindows XP/2000が動くレベルであれば問題ありません．ただし，低価格のダウンロード・ケーブルはパソコンのパラレル・ポートを使用します．最新のノート・パソコンにはパラレル・ポートが付いていないので，注意してください．ダウンロード・ケーブルはFPGAベンダの販売代理店などから購入可能です．

● FPGA開発ツールやシミュレータなどをインストール

まず，設計で使用するツールをそろえます．今回は費用をかけずに試せるように，ツールはWindows上で動作する無償ツールを使って設計を進めていきます．FPGA開発ツールは，今回使用するFPGAの供給元である米国Xilinx社が無償で提供している「ISE WebPACK」を使用します．ISE WebPACKはHDLコードの論理合成，配置配線，FPGAへの回路情報のダウンロードなどの機能を備えています．シミュレータはオープン・ソースとして公開されている「IcarusVerilog」を使用することにします．波形表示ツールにはオープン・ソースの「GTKWave」を使用します．

1）ISE WebPACKの導入

本誌2005年1月号には，ISE WebPACKが収録されたCD-ROMが付属しています．ここからインストールすれば，ISE WebPACKを利用できます．

1

インターネット経由でISE WebPACKをダウンロードする場合は，Xilinx社の日本支社のサイト（http://www.xilinx.co.jp/）から「製品とサービス」→「ISEデザインツール」→「ISE WebPACK」の順に移動してユーザ登録を行い，「Xilinx Web Install」をクリックして必要なファイルをダウンロードします．ダウンロードしたファイルを実行し，指示に従ってインストールしてください．

次に，作業用のプロジェクトを作成します．デスクトップ上の「Project Navigator」をダブルクリックしてISE WebPACKを起動します．メニューから「File」→「New Project」を選択してNew Projectダイアログを開きます．New Projectダイアログに設定する内容は，使用するFPGAに合わせます．今回の設定内容は**表1**になります．

プロジェクトにソースを登録します．本誌付属のCD-ROMに収録されているsample.vを「Project Location」で指定したディレクトリにコピーします．次に，メニューから「Project」→「Add Source ...」を選択して，Add Existing Sourceダイアログでsample.vを選択し，Choose Source Typeダイアログで「Verilog Design File」を選択します．**図3**がsample.vを登録したProject Navigatorウィンドウです．左上の「Source in Project」に登録されているファイルが表示されます．その下のProcesses for Sourceウィンドウには実行するツールが表示されています．ツールは工程順に上から並んでいます．前の工程に未実行ツールがある場合，自動的に未実行ツールが実行されます．実行済みのツールには緑のチェック・マークが付きます．

注1：gEDAとは，GNUプロジェクトの提唱するGPL（General Public License）に準拠したフリーのEDAツールを開発するプロジェクト．さまざまなEDAツールが公開されている．

表1　New Projectダイアログの設定内容

1ページ目	
Project Name	任意プロジェクト名
Project Location	任意ディレクトリ
Top-Level Module Type	HDL
2ページ目	
Device Family	Spartan3
Device	xc3s50
Package	vq100
Speed Grade	-4
Synthesis Tool	XST（VHDL/Verilog）
Simulator	Other
Generated Simulation Language	Verilog
3～5ページ目	
各項目	デフォルト設定

2）IcarusVerilogの導入

IcarusVerilogは，gEDA[注1]に登録されたオープン・ソースのVerilog HDLシミュレータです．オープン・ソースというと自分でコンパイルする必要があり，敬遠したくなりますが，Windows版のIcarusVerilogはインストーラが用意されているので簡単に導入できます．

IcarusVerilogはインターネットのWebサイト（http://icarus.com/eda/verilog/）からダウンロードします．ftp://ftp.icarus.com/pub/eda/verilog/v0.8/Windows/にあるiverilog-0.8-setup.exeをダウンロードして実行してください．

インストールが完了したら試しにシミュレーションを実行してみます．本誌付属のCD-ROMから任意のディレクトリへsample.vとsample_test.vをコピーします．コマンド・プロンプトを起動して，ファイルをコピーしたディレクトリに移動します．

まずは，シミュレーション用のオブジェクトを作成します．

```
>iverilog sample.v sample_test.v
```

オブジェクト名はデフォルトではa.outになります．次にオブジェクトを実行します．

```
>vvp a.out
```

図4はシミュレーションを実行しているようすです．

3）GTKWaveの導入

GTKWaveもgEDAに登録されたオープン・ソースの波形ビューワです．IcarusVerilogが出力したVCD（value change dump）ファイルの波形を表示します．GTKWave

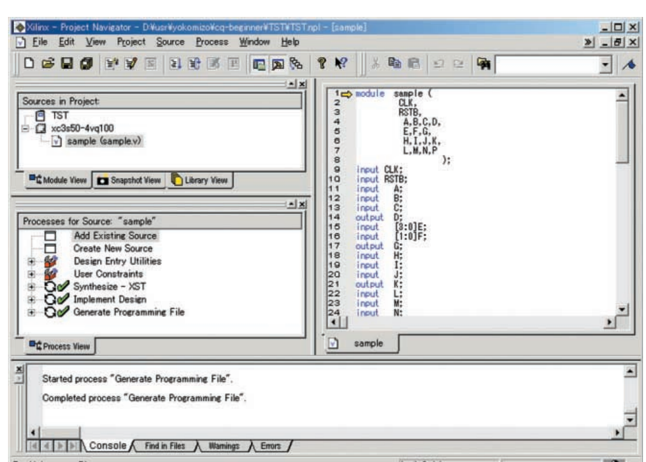


図3　Project Navigatorのようす

左上の「Source in Project」でプロジェクトやソースの管理を行う．その下の「Processes for Source」では，設計に必要なツールが工程順に並んでいる．

のWebサイトはhttp://www.cs.man.ac.uk/apt/tools/gtkwave/です．このサイトではlinux用のソース・コードを公開していますが，Windows版は別に開発されており，ここでは公開していません．Windows版はThe Win32 GTKWave Homepage（http://www.geocities.com/SiliconValley/Campus/3216/GTKWave/gtkwave-win32.html）で公開されています．

GTKWave-win32-1.3.19.zip，DLL-19990828.zip，zlib114.zipのファイルをダウンロードします．zipファイルを解凍して，ファイルをすべて同じディレクトリに入れます．ファイルを入れたディレクトリをシステム環境変数のpathに追加します．「コントロールパネル」→「システムのプロパティ」→「詳細設定」→「環境変数」を選択し，システム環境変数のpathへ解凍したファイルを置いたディレクトリを追加します．

次にGTKWaveを起動してみます．iverilogを実行したディレクトリを移動して，

```
>winwave sample_test.vcd
```

と入力し，シミュレーションで作成したvcdファイルを表示します．**図5**は波形を表示したGTKWaveのようすです．

● ハードウェアを設計するための言語を使う

HDLはその名のとおりハードウェアを設計するために作られた言語です．HDLとしてはVerilog HDLとVHDLの二つが普及しています．これらの言語には，ハードウェアの動作を表現するために通常のプログラミング言語にはない特徴があります．いちばん大きな特徴は，コンカレントな（並行）動作を表現できることです．通常のプログラミング言語は，プログラムを先頭から順番に処理するシーケンシャルな動作を表現します．一方，HDLでは複数の処理を並行して実行するコンカレントな動作を表現できます注2．ただし，個別の処理の内部ではシーケンシャルな動作も可能です．

では，現在，広く使われているVerilog HDLとVHDLのどちらを先に覚えたらよいのでしょうか．文法に違いがあっても，回路を表現するための考えかたは同じです．どちらか一方の言語で回路設計を習得すれば，もう一方への移行は容易です．ただし，VHDLにはパッケージやコンフィグレーションなど，敷居の高いしくみが盛り込まれているので，身近にVHDLに詳しい先輩がいない場合や独学の場合は，Verilog HDLから始めることをお勧めします．

実際に設計する場合には，客先からの要求や設計資産の有無，手持ちのツールの対応言語などを考慮して使用する言語を決定することになります．

● 組み合わせ回路と順序回路では記述方法が異なる

HDLの文法を習得しただけでは残念ながら回路を作ることはできません．回路を設計するためには，論理合成が可能な記述スタイルを覚える必要があります．部品の接続が記述されたネットリストは，**図6**のように論理合成ツールを利用してHDLコードから作成します．ネットリストを得るためには，論理合成ツールが回路を推定できるHDL

注2：シミュレータは，実際に処理を並行に実行しているわけではない．並行実行の処理をシミュレーションによって模擬しているだけである．

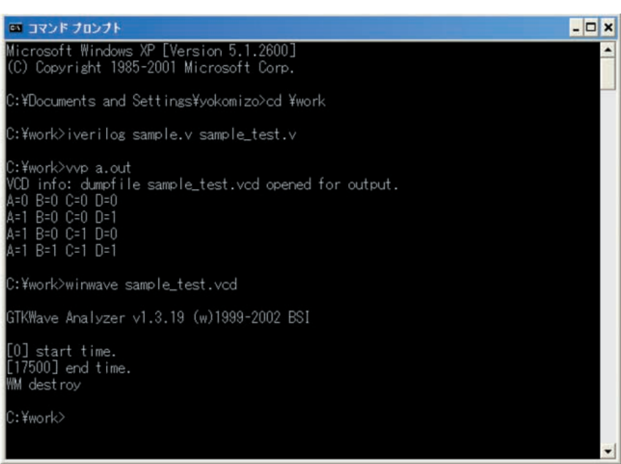


図4 IcarusVerilogの実行のようす
iverilogコマンドで実行オブジェクトを作成する．vvpコマンドでオブジェクトを実行する．「A=0 B=0 C=0 D=0」はテストベンチの値の表示．winwaveコマンドは波形表示ツール「GTKWave」の実行コマンド．

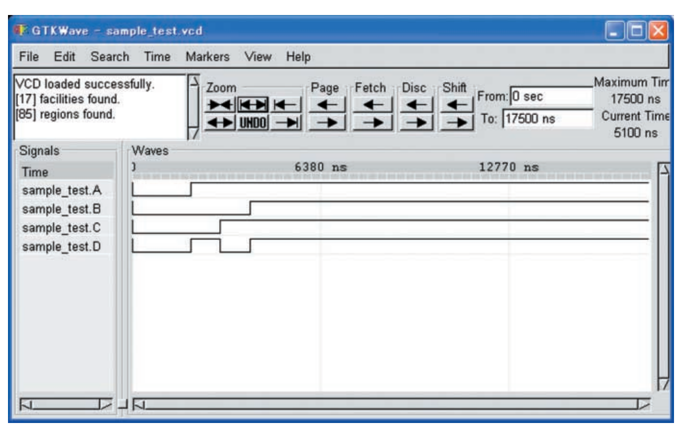


図5 GTKWaveの画面例
シミュレーションで作成した波形ファイル（sample_test.vcd）を表示している．リスト1の2 to 1セレクタのシミュレーション波形．

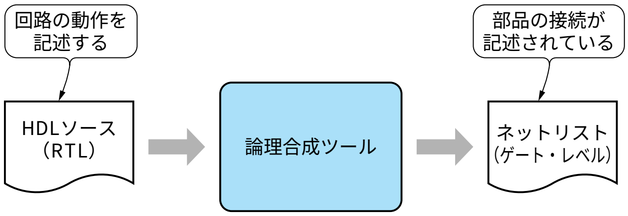


図6　論理合成ツールの入力と出力
論理合成ツールは，回路の動作が記述されているHDLソースをネットリストに変換する．論理合成ツールに渡すHDLソースは記述スタイルが決められている．

リスト1
assign文で記述した2 to 1セレクタ

```
assign D = (C==1'b1) ?B:A;
```

リスト2
always文で記述した4 to 1セレクタ

```
always@(E or F)
//EかFが変化したら実行
  begin
    case (F)
      2'b00:G<=E[0];
      2'b01:G<=E[1];
      2'b10:G<=E[2];
      2'b11:G<=E[3];
    endcase
  end
```

コードになっている必要があります．

この回路を推定できるスタイルをRTL記述と呼んでいます．DFF（Dフリップフロップ）を含まない組み合わせ回路とDFFを含む順序回路では記述方法が違っています．HDLコードを書く場合は，この違いを意識する必要があります．

1）組み合わせ回路の記述

組み合わせ回路を表現する方法には，assign文を使う方法とalways文を使う方法があります．簡単な組み合わせ回路はassign文で表現できます．**リスト1**はassign文で記述した2 to 1セレクタです．assign文は右辺の信号値に変化が発生すると演算を実行するので，組み合わせ回路の動作と一致します．「Cが1'のときにBを出力，それ以外はAを出力」となります．

リスト2は4 to 1セレクタをalways文で記述しています．always文ではif文やcase文を使って論理を記述できるので，assign文より複雑な組み合わせ回路を表現できます．ただし，組み合わせ回路をalways文で表現するための決まりごとがあります．組み合わせ回路は入力信号が変化すると新しい出力が決まるので，alwaysの後の()内のセンシティビティ・リスト注3には入力信号をすべて書きます．そして，beginとendの間に処理を書きます．組み合わせ回路の出力に相当する信号にかならず代入を実行してください．代入が実行されない条件があった場合，信号の値が保持されるため，論理合成によってラッチが生成されてしまいます．

リスト3はfunction文を使った例です．function文はassign文やalways文からコールして使用します．同じ処理が複数存在するときは，function文を利用すると効率的に設計できます．

2）順序回路の記述

順序回路はalways文で記述しますが，やはり決まったスタイルがあります．always文で記述できる順序回路は，**図7**のようにDFFのみでもよいし，組み合わせ回路を含めることもできます．ただし，出力はDFFに限定されます．**図8**のように順序回路のalways文から組み合わせ回路の信号を出力する構成は，うまく論理合成できません．**リスト4**は，2 to 1セレクタ付きのDFFを記述したalways文です．always文のセンシティビティ・リストには，エッジを指定したクロック信号と非同期のセット/リセット信号を書きます．立ち上がりはposedgeを，立ち下がりはnegedgeを指定します．出力がDFFの出力に限定されており，クロック信号と非同期セット/リセット信号がアク

注3：リストされた信号に変化があると，always文が実行される．assign文は，右辺にある信号のすべてに対してセンシティブである．

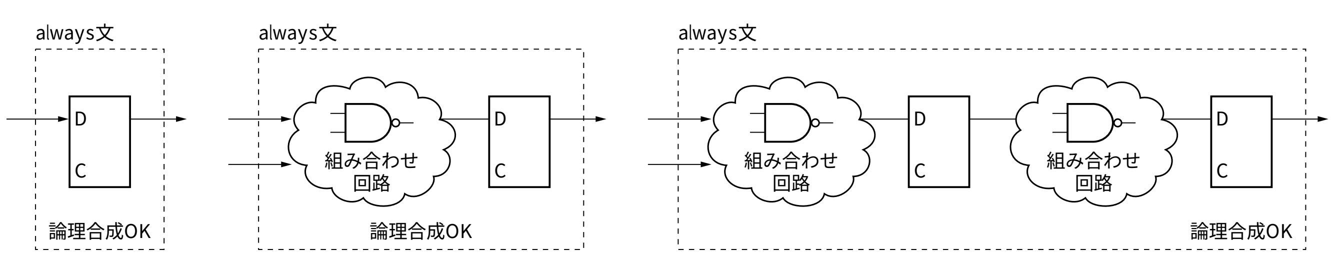


図7　論理合成可能な順序回路の構成
論理合成可能な順序回路の出力はDFF（Dフリップフロップ）になる．順序回路の記述スタイルを守ると，このような回路構成になる．

リスト3　function 文の利用例

```
function SEL_FUNC ;       //ファンクションの定義
  input IN_1;
  input IN_2;
  input SEL ;
  begin
    if (SEL==1'b1)
      SEL_FUNC = IN_1;
    else
      SEL_FUNC = IN_2;
  end
endfunction

assign K=SEL_FUNC(I,H,J);
               //ファンクションをコールして機能を実現
```

リスト4　2 to 1セレクタ付き DFF の always 文

```
always@(posedge CLK or negedge RSTB)
             //CLKの立ち上がりまたはRSTBの立ち下がりで実行
  begin
    if (RSTB==1'b0)//非同期セット・リセットの条件
      P <= 1'b0;    //非同期セット・リセットの動作
    else            //ここ以降にDFFの値を決める動作を書く
    if (N==1'b1)
      P <= M;
    else
      P <= L;
end
```

ティブになるエッジで演算を開始することで，順序回路の動作を表現できます．begin と end の間に処理を書きますが，非同期セット/リセットがある場合は制御が優先されるので，最初に条件を記述します．その次に通常の処理を書き込みます．ここで DFF になる信号にはノン・ブロッキング代入文(<=)で値を代入します．ノン・ブロッキング代入文は，各代入文の右辺の処理がすべて終了してから代入が実行されます．クロック同期で動作しているときにデータの筒抜けが発生しないため，代入の順序に関係なく DFF としてふるまいます．

ここで紹介したのは，Verilog HDL の記述スタイルのほんの一部です．詳しくは，専門の書籍などを参照してください．

● 論理合成してみよう

では，リスト1～リスト4を論理合成してみましょう．

Project Navigator をダブルクリックして，ISE WebPACK を起動します．ウィンドウ左上の「Source in Project」から sample.v を選択します．その下の Process for Source ウィンドウで「Synthesize XST」を選択して，マウスの右ボタンを押して「run」を選択すると，論理合成が実行されます．

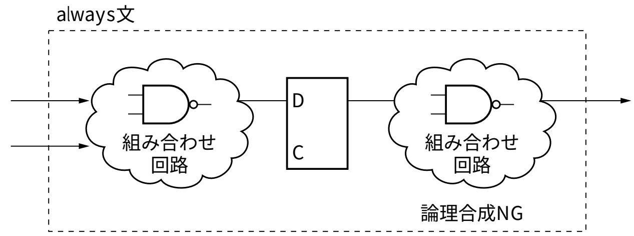


図8　論理合成できない順序回路の構成
順序回路の always 文から DFF 以外の信号を出力させることはできない．組み合わせ回路の信号を出力したい場合は，組み合わせ回路を分離して記述する必要がある．

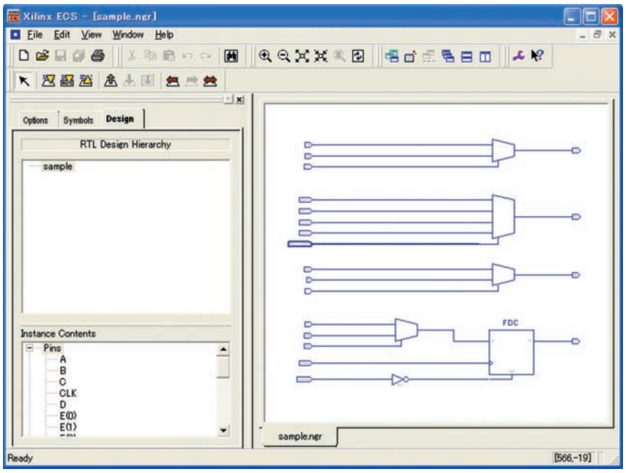


図9　ISE WebPACK で合成した回路を表示したようす
上からリスト1，リスト2，リスト3，リスト4の順で回路が並んでいる．

実行が終了して「Synthesize XST」の前に緑のチェックが付けば，論理合成は成功です．黄色の！があった場合は警告メッセージがあります．赤い×の場合はエラーです．エラー・メッセージはいちばん下のウィンドウに表示されます．

ISE WebPACK には，論理合成によって生成された回路を表示する機能があります．「Synthesize XST」の前の＋をクリックすると，「Synthesize XST」の下に項目が表示されます．その中の「View RTL Schematic」を実行すると，図9のように回路図が表示されます．自分の書いたソースがどのような回路になるかを知ることは，RTL 記述の理解に役立ちます．これから HDL 設計を始められる方は，この機能を活用してみてください．

2 演算当てゲームを作る

ここでは，FPGA 基板を利用して簡単な“演算当てゲー

ム”を作ります．ランダムに生成した式の計算結果と計算に使用した数字を表示して，式に使用した演算（＋，－，×）を当てるゲームです．

● 仕様を検討する

動作の概要は図10のようになります．SW4というスイッチを押すとゲームが始まります．液晶ディスプレイ（LCD）には四つの数字と計算結果が示されます．三つの演算は，初めは“？”と表示されます．SW1～3を押すたびに，演算は“＋”→“－”→“＊”→“＋”の順で変化し，三つの演算が正しい場合に正解になり，次の問題に移ります．5問正解したら経過時間を表示して終了です．

動作概要をもとに入出力信号を定義します．表2が端子表です．

● 部品を入手する

仕様が決まったので，必要なハードウェアを準備をします．FPGAについては，前述のとおり本誌2005年1月号に付属したFPGA基板（Spartan-3 XC3S50を搭載）を利用します．この基板に接続するサブボードも作成します．図11はサブボードの回路図で，表3は部品の一覧表です．

入力信号はスイッチで生成します．出力はLCDモジュールになります．クロック発生用に水晶発振器を使用します．水晶発振器は8MHzのものを使用していますが，HDLソースを書き換えれば別の周波数のものでも使用可能です．筆者の場合，FPGA基板以外の部品は秋月電子通商（http://akizukidenshi.com/catalog/）で購入しました．

● ハードウェアを作成する

まずはFPGA基板に部品を取り付けます．3端子レギュレータは，FPGA基板の電源を5V対応にするために必要です．サブボードとの接続に使用する34ピンと40ピンのピン・ヘッダについては，カット可能なピン・ヘッダから必要なピン数を切り取って使用しています．1列のピン・ヘッダは電源ピン用とダウンロードに使用するJTAGピン用です．スイッチはリセット用です．写真1は部品の取り付けが完了したFPGA基板のようすです．FPGA基板の詳細については，本誌2005年1月号の特集記事を参照してください．

次に，図11に従ってサブボードに部品を取り付けます．写真2は完成したサブボードのようすです．34ピンと40ピンのピン・ソケットは，FPGA基板のピン・ヘッダの位置に合わせて取り付けてください．LCDモジュールのソケットも，LCDモジュールとFPGA基板がぶつからないように位置を決めます．

サブボードが完成したら配線の確認を行います．FPGA基板とLCDモジュールを接続して完成です．5V電源を接続して可変抵抗を動かし，LCDモジュールのコントラストを調整してください．

● ブロック分割を行う

ハードウェアの用意ができたので，FPGA設計の話に戻ります．仕様は決まったのですが，いきなりHDLを記述するにはまだ無理があります．機能別にいくつかのブロックに分割して，作業を進めやすくします．ブロック分割は，

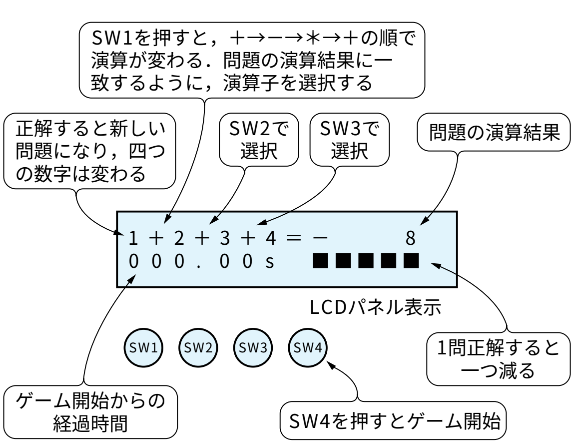


図10　作成するゲームの概要

スタート前に数字をx，演算を？で表示する．5問正解すると，経過秒数の表示が停止する．

表2　作成する回路の端子表

信号名	入出力	内　容
CLK	I	入力クロック（8MHz）
RSTB	I	リセット信号．“L”でリセット
SW1	I	演算1の選択．push（“H”→“L”）で次の演算
SW2	I	演算2の選択．push（“H”→“L”）で次の演算
SW3	I	演算3の選択．push（“H”→“L”）で次の演算
SW4	I	push（“H”→“L”）でスタート/ストップ リセット解除時に“L”の場合は，デバッグ・モードに移行する
E	O	LCDモジュール・アドレス・イネーブル
RW	O	LCDモジュール・リード/ライト指定 “H”でリード，“L”でライト
RS	O	LCDモジュール・レジスタ・セレクタ
D[7:0]	O	LCDモジュール・データ・バス
LED1	O	“L”で点灯．SW1～4を押すと点灯
LED2	O	“L”で点灯．1秒周期で点滅

以下の点に注意します．

- **独立した機能で1ブロックとする**
- **複数使用する回路を1ブロックにして，流用することで設計の効率を上げる**
- **再利用できそうな部分と今回のみ使用する機能を分ける**

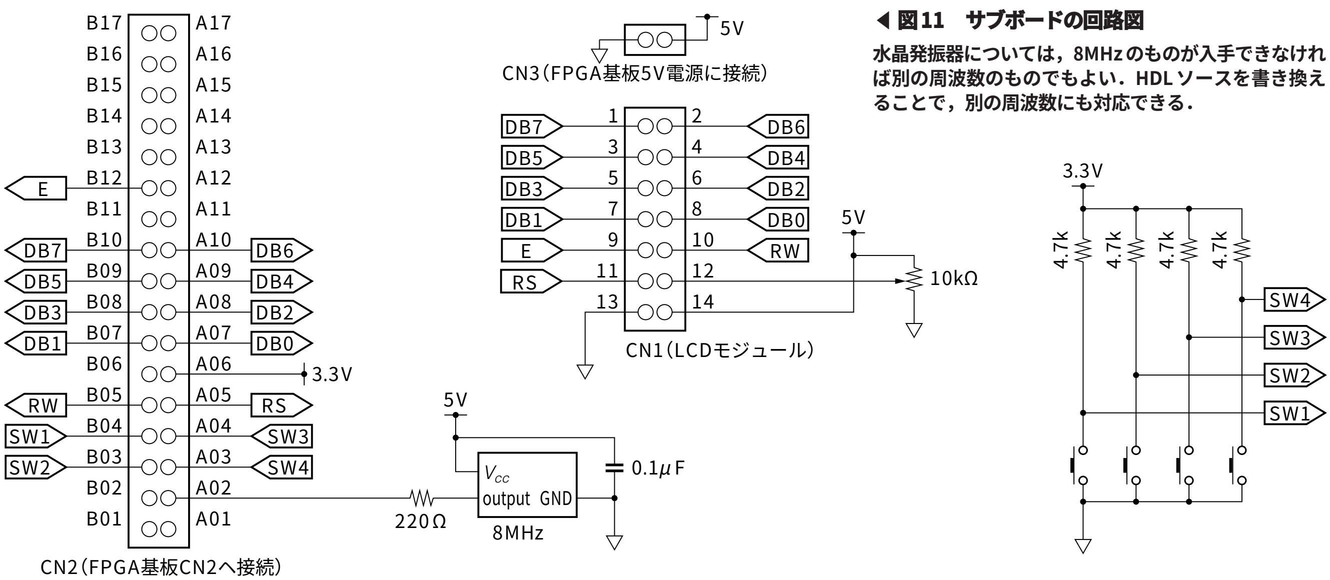
次にブロック間の接続がわかるようにブロック図を作ります．各ブロックの機能を表4に，ブロック図を図12に示します．SW_IFとCNT_DECは複数回使用するため，それぞれ一つのブロックになっています．また，今回のゲームに固有の機能はMAINに持たせ，外部インターフェース用のブロック(SW_IF，LCD_CTRL)については別の機能を設計するときに流用できるように独立させました．

● 各ブロックをHDLでコーディング

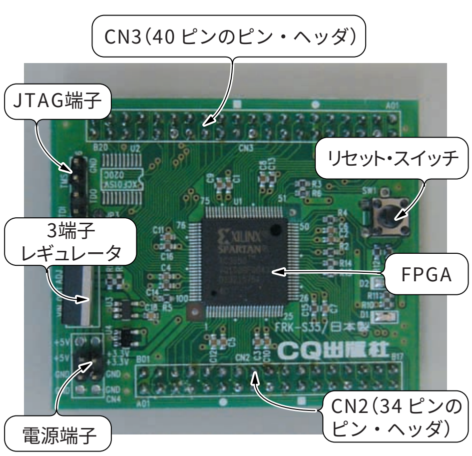
では，各ブロックをコーディングしていきます．一つの

表3 部品の一覧表

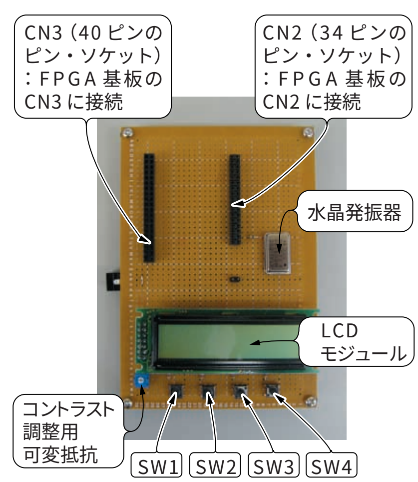
部品名	個数	内 容
FPGA基板	1	XC3S50搭載基板．本誌2005年1月号に付属
LM317T	1	3端子レギュレータ 5V電源対応のため，FPGA基板に実装
34ピンのピン・ヘッダ	1	FPGA基板に実装．サブボード接続用
40ピンのピン・ヘッダ	1	FPGA基板に実装
1列ヘッダ	8	FPGA基板電源用．JTAG用(ダウンロード)
ユニバーサル基板	1	サブボードの基板用．1.6t× 95mm × 138mm
34ピンのピン・ソケット	1	サブボードに実装．FPGA基板接続用
40ピンのピン・ソケット	1	サブボードに実装．FPGA基板接続用
スイッチ	5	小型プッシュ・スイッチ．FPGA基板のリセット用×1，サブボードのスイッチ×4
LCDモジュール	1	SC1602BS＊B．秋月電子通商で取り扱い コネクタが付属する
可変抵抗	1	10kΩ．LCDモジュールのコントラスト調整用
水晶発振器	1	8MHz (8MHz以外でもかまわない)
抵抗220Ω	1	1/6Wカーボン抵抗
抵抗4.7kΩ	4	1/6Wカーボン抵抗
5V電源	1	ACアダプタ(安定化されているもの)
DCジャック	1	ACアダプタ用のDCジャック サブボードに実装



◀ 図11 サブボードの回路図
水晶発振器については，8MHzのものが入手できなければ別の周波数のものでもよい．HDLソースを書き換えることで，別の周波数にも対応できる．



◀ 写真1
部品を取り付けたFPGA基板
3端子レギュレータの向き，および電源端子に3.3V用と5V用があることに注意する必要がある．



▶ 写真2
部品を取り付けたサブボード
34ピンのピン・ソケットが入手できなかったので，カット可能なピン・ソケットを34ピンにカットして使用している．40ピンのピン・ソケットには何も配線していない．

表4　各ブロックの機能

ブロック名	機　能
SW_IF	SWからの入力のチャタリング対策と立ち下がり検出
LCD_CTRL	LCDモジュールの初期化・表示制御
TIMER	秒数のカウント
CNT_DEC	TIMERのサブモジュール．10進カウンタ
MAIN	ゲーム固有の機能
MAIN_CTRL	MAINのサブモジュール．スタート/ストップの制御
CALC	MAINのサブモジュール．問題の作成と解答のチェック

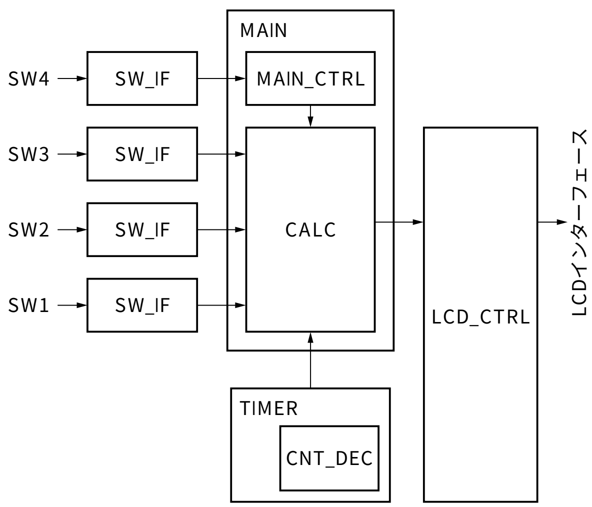


図12　ブロック図
SW_IFは，同じブロックを四つ使用する．ゲーム固有の機能はMAINに集めた．MAINを変更することで，別のゲームも作れる．

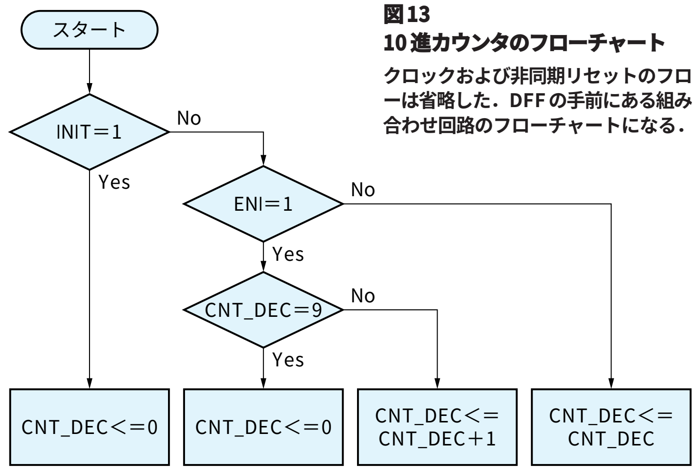


図13
10進カウンタのフローチャート
クロックおよび非同期リセットのフローは省略した．DFFの手前にある組み合わせ回路のフローチャートになる．

ブロックがVerilog HDLのmoduleに相当します．最初に共通で使用するモジュールを作成します．まずは，10進カウンタ(CNT_DEC.v)から作ります．ISE WebPACKには新規にソース・ファイルを作成する際の補助機能があるので利用します．Project Navigatorのメニューから「Project」→「New Source」を選択して，New Sourceダイアログを開きます．1ページ目では，「File Name」へ作成するモジュール名を入力して，右側で「Verilog Module」を選択します．2ページ目では入出力信号を指定します．3ページ目で[完了]ボタンをクリックすると，入出力信号だけが定義されたソースが作成されます．このソースに動作を書き込みます．動作を書く前のフローチャートで10進カウンタの動きを整理します．**図13**がフローチャートです．条件判断がif文になると考えてソースを書いてください．**リスト5**は10進カウンタの記述です．カウンタは，always文を使った順序回路になっています．イネーブル出力は，assign文の組み合わせ回路になっています．

リスト5　10進カウンタ

```
module  CNT_DEC (CLK,RSTB,
                 INIT,ENI,ENO,COUNT_DEC     );
   input    CLK; // 8MHz 0.125us
   input    RSTB;
   input    INIT;
   input    ENI;
   output   ENO;
   output   [3:0]COUNT_DEC;
   reg      [3:0]COUNT_DEC;
   always@(posedge CLK or negedge RSTB)
   begin
     if (RSTB==1'b0)
       COUNT_DEC <= 4'd0;
     else
       if (INIT==1'b1)
         COUNT_DEC <= 4'd0;
       else
         if (ENI==1'b1)
           if (COUNT_DEC == 4'd9)
           COUNT_DEC <= 4'd0;
           else
             COUNT_DEC <= COUNT_DEC + 4'd1;
         else
           COUNT_DEC <= COUNT_DEC ;
   end
   assign ENO = ((ENI==1'b1)&&(COUNT_DEC==4'd9))?1'
                                             b1:1'b0;
endmodule
```

● 状態遷移図からソースを作る

ISE WebPACKには，状態遷移図からHDLを生成する「StateCAD」というツールが用意されています．SW_IFを状態遷移図から作ってみます．SW_IFでは，スイッチからの入力が"H"から"L"へ変化したことを検出しています．スイッチからの入力では，信号変化時に"H"，"L"がばたつくチャタリングが発生します．チャタリングによる誤動作を防止するため，間隔を置いて同じ値が2回続いたときに値を確定します．StateCADを使って，この動作を状態遷移図にしてみましょう．

「Project」→「New Source」を選択して，New Sourceダイアログの1ページ目で「File Name」へ作成するモジュ

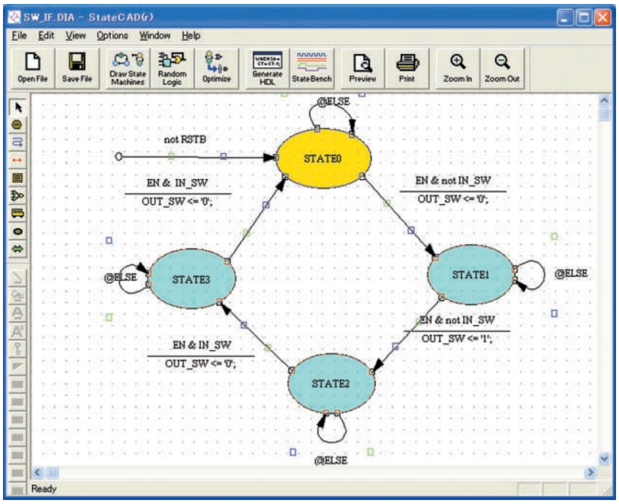


図14 StateCADで描いたSW_IFの状態遷移図
リセット後，STATE0からスタートする．EN＝1かつIN_SW＝0の条件で，STATE1に移動する．さらに，EN＝1かつIN_SW＝0になると，STATE2に移動する，このときにOUT_SW＝1にする．

ール名を入力し，左側で「State Diagram」を選択すると，StateCADが起動します．状態遷移図の描きかたはStateCADのヘルプを参照してください．操作は難しくありません．状態遷移図を描いた後，［Generate HDL］ボタンをクリックすると，Verilog HDLのソースが生成されます．**図14**はStateCADで描いたSW_IFの状態遷移図です．EN＝1のときの入力信号が2回連続して“L”のときに，SW_SIGを‘1’にします．**リスト6**はStateCADが生成した記述です．順序回路のalways文で代入文がブロッキング代入になっています．ほかのモジュールと統一するため，ノン・ブロッキング代入に修正してあります．

● MAINにゲーム固有の機能を集めた

各ブロックの機能について説明します．

1）LCDモジュールの制御

LCDモジュールは使用する前に初期化が必要です．初期化の手順は**図15**の左側のようになります．初期化終了後に表示データをLCDモジュールに送ります．初期化終了後は，DB7をリードすることでLCDモジュールのbusy状態を確認できます．今回はLCDモジュールの動作タイミングがわかっているので，busy状態を確認せずに一定の周期で書き込みを行っています．

2）メインのコントローラ

ブロックMAINには，このゲーム固有の機能を集めてあります．MAINには二つのモジュールがあり，MAIN_

リスト6 StateCADが生成したSW_IFの記述

```
module sw_if(CLK,EN,IN_SW,RSTB,OUT_SW);

  input CLK;
  input EN,IN_SW,RSTB;
  output OUT_SW;
  reg OUT_SW,next_OUT_SW;
  reg SV0,next_SV0,SV1,next_SV1;

  always @(posedge CLK or negedge RSTB)
  begin
    if ( ~RSTB ) begin
      /*      ブロッキング代入
      SV0 = 0;
      SV1 = 0;
      OUT_SW = 0;
      */
      SV0 <= 0;  //ノン・ブロッキング代入
      SV1 <= 0;
      OUT_SW <= 0;
    end else
    begin
      /*      ブロッキング代入
      SV0 <= next_SV0;
      SV1 <= next_SV1;
      OUT_SW <= next_OUT_SW;
      */
      SV0 <= next_SV0;   //ノン・ブロッキング代入
      SV1 <= next_SV1;
      OUT_SW <= next_OUT_SW;
    end
  end

  always @ (EN or IN_SW or SV0 or SV1)
  begin

    if ( EN & ~IN_SW & SV1 | SV0 & ~SV1 | ~EN & SV0
                                        | ~IN_SW & SV0 )
      next_SV0=1;
    else next_SV0=0;

    if ( EN & ~IN_SW & ~SV0 & ~SV1 | IN_SW & ~SV0 &
        SV1 | EN & IN_SW & SV0 & ~SV1 | ~EN & SV1 |
                 ~IN_SW & SV0 & SV1 ) next_SV1=1;
    else next_SV1=0;

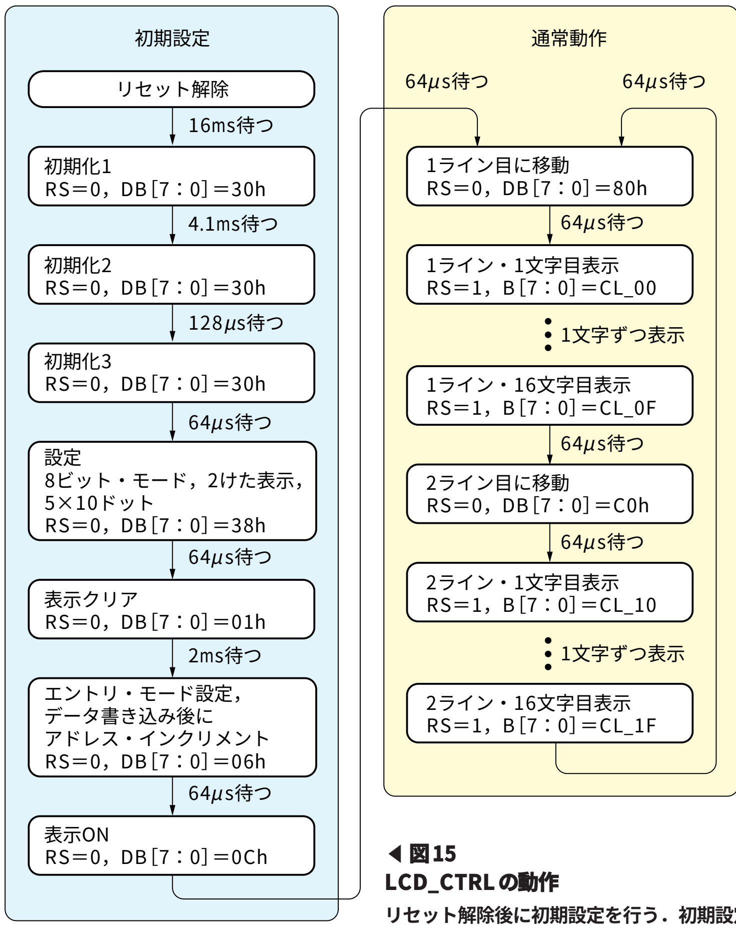
    if ( EN & ~IN_SW & ~SV0 & SV1 ) next_OUT_SW=1;
    else next_OUT_SW=0;
  end
endmodule
```

CTRLはコントローラ，CALCは問題の出題と解答のチェックを行います．**図16**はMAIN_CTRLの状態遷移図です．状態は停止と出題の二つがあり，出題状態の中には問題ごとに状態を持っています．問題ごとの状態は正解のたびに一つ移動するだけなので，カウンタ回路にしてあります．

3）問題の答えを求める演算回路

次に，このゲームの心臓部とも言える問題の答えを求める演算回路（CALC.v）を作ります．やりたいことはランダムに選択された四つの数字と三つ演算子からなる式の計算です．掛け算は優先して演算する必要があり，演算子は毎回変更されるので処理を書くのは難しそうです．このようなデータの演算を行う回路は，**図17**のようなデータ・フロー図で表して処理手順を明確にします．演算方法は，手で計算する場合の算術演算の手順をモデル化しています．

1



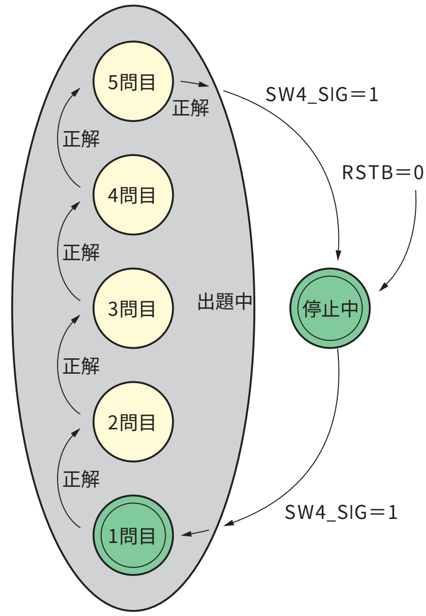
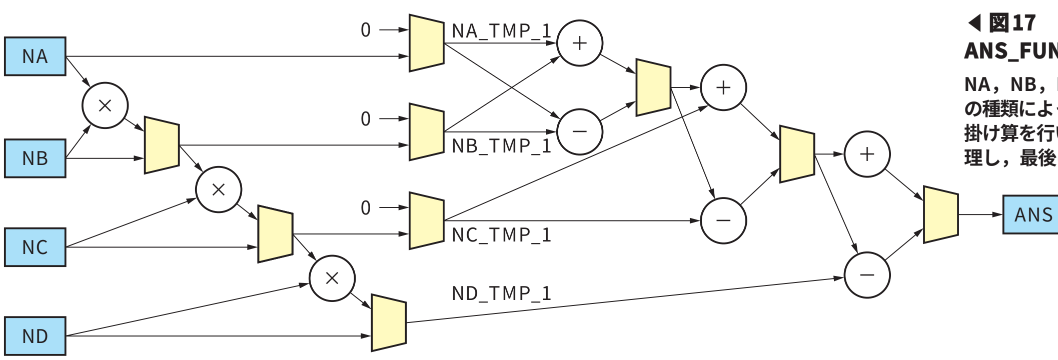


図16　MAIN_CTRLの状態遷移図

出題状態でSW4が押された場合，問題数にかかわらず停止状態に遷移する．問題数の遷移は正解のたびに一つ移動するだけなので，カウンタで作ってある．

◀ **図15**
LCD_CTRLの動作

リセット解除後に初期設定を行う．初期設定後は文字データを繰り返し送る．



◀ **図17**
ANS_FUNCのデータ・フロー

NA，NB，NC，NDは入力する数字．演算の種類によってセレクタを制御する．初めに掛け算を行い，次に掛け算で0になる項を処理し，最後に足し算と引き算を行う．

前半部分では掛け算を行い，中間部分では掛け算で不要になった項を0にします．後半部分では足し算と引き算を行って解答を算出しています．

リスト7はこの演算を記述したfunction文です．出題と解答のチェックの2ヵ所で演算が必要なので，function文で記述しています．

● 論理合成・配置配線を試行する

ここまでで紹介したコード以外のソースは本誌付属のCD-ROMに収録されています．**図18**の階層図を参考にして，ソース・コードをプロジェクトに追加して設計を完成させます．設計が完了したのでシミュレーションで論理検証を始めたいところですが，うまく論理合成や配置配線が行えないソース・コードだと困るので，まずは論理合成・配置配線を試行してみます．残念なことにISE WebPACKの実行結果はエラーになります．ログを解析して対策を考えることになります．

ログを解析すると，XC3S50には乗算器が四つ用意され

リスト7　演算回路(CALC.v)を記述したfunction文

```
function [15:0]ANS_FUNC ;
  input [1:0]SA;
  input [1:0]SB;
  input [1:0]SC;
  input [3:0]NA;
  input [3:0]NB;
  input [3:0]NC;
  input [3:0]ND;
  reg [3:0]NA_TMP_1;
  reg [7:0]NB_TMP_0;
  reg [7:0]NB_TMP_1;
  reg [11:0]NC_TMP_0;
  reg [11:0]NC_TMP_1;
  reg [15:0]ND_TMP_1;
  reg [1:0]SA_TMP;
  reg [1:0]SB_TMP;
  reg [1:0]SC_TMP;
  reg [16:0]ANS_A;
  reg [16:0]ANS_B;
  reg [16:0]ANS_C;
  reg [15:0]ANS;
  begin
  //掛け算・演算の変更
  if (SA==2'b10)
    begin
      NB_TMP_0 = NA * NB ;
        SA_TMP = 2'b00;
    end
  else
    begin
      NB_TMP_0 = {4'd0,NB} ;
        SA_TMP = SA;
    end
  if (SB==2'b10)
    begin
      NC_TMP_0 = NB_TMP_0 * NC ;
        SB_TMP = SA_TMP;
    end
  else
    begin
      NC_TMP_0 = {8'd0,NC} ;
        SB_TMP = SB;
    end
  if (SC==2'b10)
    begin
      ND_TMP_1 = NC_TMP_0 * ND ;
        SC_TMP = SB_TMP;
    end
  else
    begin
      ND_TMP_1 = {12'd0,ND} ;
        SC_TMP = SC;
    end
  //掛け算で不要になった項を0にする
  if (SA==2'b10)
    begin
      NA_TMP_1 = 4'd0;
    end
  else
    begin
      NA_TMP_1 = NA;
    end
  if (SB==2'b10)
    begin
      NB_TMP_1 = 8'd0;
    end
  else
    begin
      NB_TMP_1 = NB_TMP_0;
    end
  if (SC==2'b10)
    begin
      NC_TMP_1 = 12'd0;
    end
  else
    begin
      NC_TMP_1 = NC_TMP_0;
    end
  //足し算・引き算
  if (SA_TMP==2'b01)
    ANS_A = {12'd0,NA_TMP_1} - {8'd0,NB_TMP_1};
  else
    ANS_A = {12'd0,NA_TMP_1} + {8'd0, NB_TMP_1};

  if (SB_TMP==2'b01)
    ANS_B = ANS_A - {4'd0,NC_TMP_1};
  else
    ANS_B = ANS_A + {4'd0,NC_TMP_1};

  if (SC_TMP==2'b01)
    ANS_C = ANS_B - ND_TMP_1;
  else
    ANS_C = ANS_B + ND_TMP_1;

    ANS_FUNC = ANS_C[15:0];
  end
endfunction
```

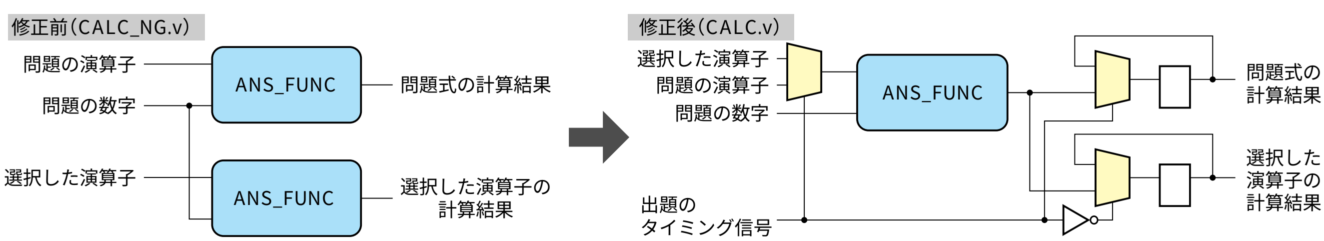


図19　ANS_FUNCの修正

修正前は2個のANS_FUNCを使用している．修正後はANS_FUNCを一つにした．出題のタイミングで問題式の結果を算出し，それ以外のときはスイッチで選択した演算を算出している．

ていますが，この設計では五つの乗算器を使用するため，演算器が足りないようです．乗算器は数式の演算を行うファンクションANS_FUNC内で三つ使用します．ANS_FUNCは出題と回答のチェックの2ヵ所で使用するため，ANS_FUNCに相当する回路が2個生成され，乗算器は6個必要になります．ただし，NAとNBの乗算は2ヵ所で演算内容が同じなので，論理合成時に乗算器が共有されて最終的には5個の乗算器を使用しています．

出題の演算は新しい問題に移ったときに1回実行すればよいので，つねにANS_FUNCが必要ではありません．そこで，入力信号をセレクタで切り替えて，出題用と回答のチェック用のANS_FUNCを共用して，ANS_FUNCに相

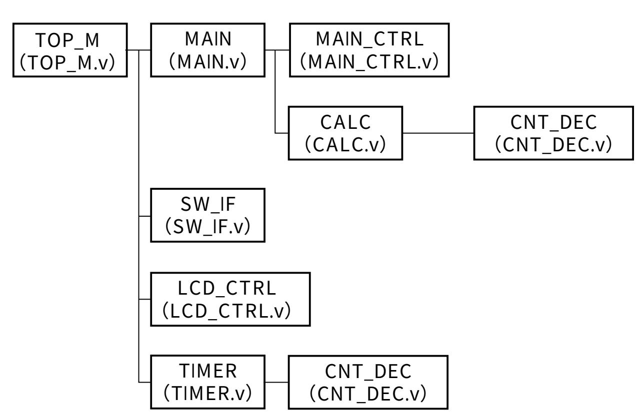


図18　階層図

モデルの最上位階層はTOP_M．CNT_DECは，CLACとTIMERでサブモジュールとして使用する．

当する回路を一つで済むように修正します．図19は修正部分の回路構成です．

● テストベンチを作成してRTLシミュレーション

次にテストベンチを作成します．テストベンチからRTLモデルに信号を入力して，RTLモデルの論理検証を行います．図20はテストベンチの構成です．下位の階層として，検証対象のRTLモデルの最上位階層のTOP_Mをインスタンスしています．テストベンチは論理合成しないので，RTLのスタイルにこだわらずに記述できます．リスト8はテストベンチの記述の一部です．

テストベンチが完成したらRTLシミュレーションを実施します．ソース・コードのあるディレクトリに移動して，IcarusVerilogを実行します．コマンドを毎回入力するのはたいへんなので，シミュレーションから波形表示までを一括して実行するバッチ・ファイルを作成することにします．リスト9は，今回使用したバッチ・ファイルです．エクスプローラからバッチ・ファイルをダブルクリックすると，シミュレーションから波形表示までが自動的に実行されます．

図21はMAIN_CTRLの波形です．これを見て，期待どおりに動いているかどうかを確認します．おかしい部分があった場合にはソース・コードを修正して，再度シミュレ

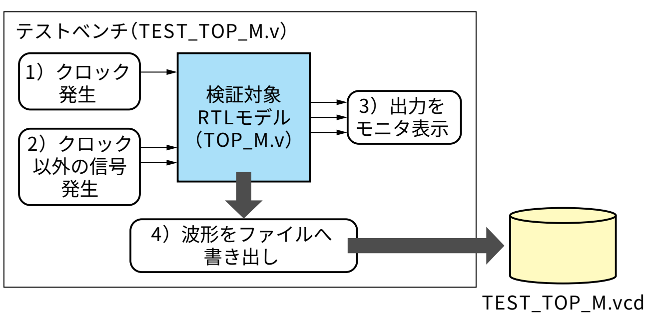


図20　テストベンチ（TEST_TOP_M.v）の構成

検証対象（TOP_M）は下位モジュールとしてインスタンスしてある．テストベンチは論理合成しないので，RTLのスタイルにこだわることなく記述できる．

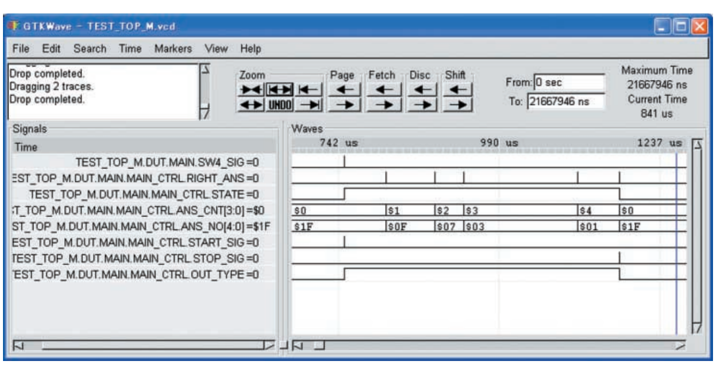


図21　MAIN_CRTLのシミュレーションの波形

図16の状態遷移図の動きを確認する．STATE＝0（停止状態）でSW4_SIG＝1（SW4が押された）になると，STATE＝1（出題状態）になる．STATE＝1（出題状態）でRIGHT_ANS＝1（正解）になると，ANS_CNT（問題数）がカウントアップされる．ANS_CNT＝4（5問目）でRIGHT_ANS＝1（正解）になると，STATE＝0（停止状態）に戻る．

リスト8　テストベンチの記述（一部）

```
`timescale 1 ns/1 ps
module  TEST_TOP_M ;
parameter P_CLK = 125;
  reg    CLK,RSTB;
  reg    SW1,SW2,SW3,SW4;
  wire   LED1,LED2;
  wire   RS,E,RW;
  wire   [7:0] DB;
// 1) クロック発生
always
  begin
    CLK <= 1'b0;
    #(P_CLK/2);
    CLK <= 1'b1;
    #(P_CLK/2);
  end
// 2) クロック以外の信号発生
initial
  begin
    RSTB <= 1'b0;
    SW4 <= 1'b0;
    SW1 <= 1'b1;
    SW2 <= 1'b1;
    SW3 <= 1'b1;
  #(P_CLK);
    RSTB <= 1'b1;

（途中省略）

    $finish;
  end
// 3) 出力をモニタ表示
initial
  $monitor("E=%b RS=%b DB=%h",E,RS,DB);
 // 4) 波形をファイルへ書き出し
initial
  begin
    $dumpfile("TEST_TOP_M.vcd");
    $dumpvars(0,TEST_TOP_M);
  end
// 5) RTLモデルのトップをインスタンス
TOP_M DUT(
  .CLK(CLK),
  .RSTB(RSTB),
  .SW1(SW1),

（途中省略）

endmodule
```

リスト9　バッチ・ファイル

```
iverilog -o sim.out SW_IF.v MAIN_CTRL.v CNT_DEC.v CALC.v TIMER.v LCD_CTRL.v
MAIN.v TOP_M.v TEST_TOP_M.v
vvp sim.out
winwave TEST_TOP_M.vcd
```

ーションを実行します．

● ピン配置を指定する

ピン配置はISE WebPACKから指定できます．まず，Sources in Projectウィンドウでトップ・モジュールのソース・コードを選択します．続いてProcesses for Sourceウィンドウの「User Constraints」の下の「Assign Package Pins」を選択します．そして，マウスの右ボタンを押して「Run」を選択し，「PACE」というツールを実行します．図22はPACEの実行画面です．GUI(graphical user interface)でピン配置を指定していきます．ピン配置の情報はトップ・モジュール名.ucfに記録されます．このファイルを直接編集して，ピン配置を指定することも可能です．リスト10はTOP_M.ucfの内容です．

● FPGAへ回路情報をダウンロード

ISE WebPACKで論理合成と配置配線を実施します．事前に論理合成・配置配線の試行が済んでいれば，ここはすんなりと終了するはずです．次に，FPGAへダウンロードするビットストリーム(回路情報)を作成します．Processes for Sourceウィンドウの「Generate Programming File」をクリックし，マウスの右ボタンを押して，「Properties」を選択します．図23のProcess Propertiesダイアログにある「Startup Options」の「FPGA Start-up Clock」で「JTAG Clock」を選択して，[OK]ボタンをクリックします．続いて「Generate Programming File」を選択して，マウスの右ボタンを押し，「Run」を選んで実行します．

ここまで終わったら，パソコンとFPGA基板をダウンロード・ケーブルで接続し，電源も接続します．次に，「Gene-

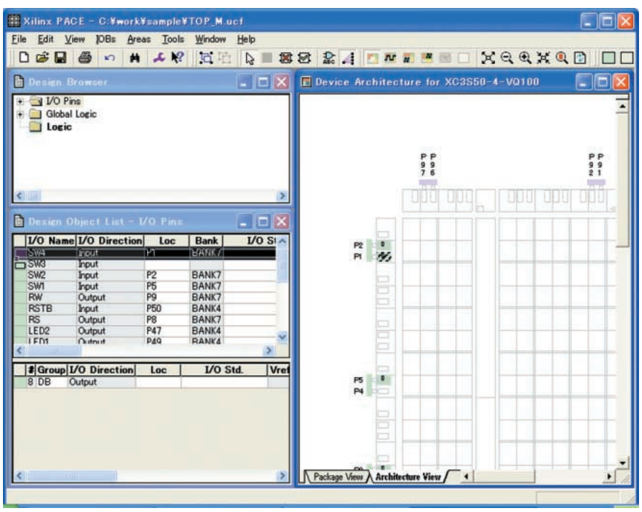


図22 PACEによるピン配置指定
ISE WebPACKのツール「PACE」を使って，ピン配置を指定している．左側に表示している端子名をマウスの左ボタンを押しながらドラックし，右側の配置図にドロップすることで，配置を指定する．

リスト10 ピン配置情報(TOP_M.ucf)

```
NET "CLK"    LOC = "P36" ;
NET "DB<0>"  LOC = "P11" ;
NET "DB<1>"  LOC = "P12" ;
NET "DB<2>"  LOC = "P13" ;
NET "DB<3>"  LOC = "P14" ;
NET "DB<4>"  LOC = "P15" ;
NET "DB<5>"  LOC = "P16" ;
NET "DB<6>"  LOC = "P17" ;
NET "DB<7>"  LOC = "P21" ;
NET "E"      LOC = "P23" ;
NET "LED1"   LOC = "P49" ;
NET "LED2"   LOC = "P47" ;
NET "RS"     LOC = "P8" ;
NET "RSTB"   LOC = "P50" ;
NET "RW"     LOC = "P9" ;
NET "SW1"    LOC = "P5" ;
NET "SW2"    LOC = "P2" ;
NET "SW3"    LOC = "P4" ;
NET "SW4"    LOC = "P1" ;
```

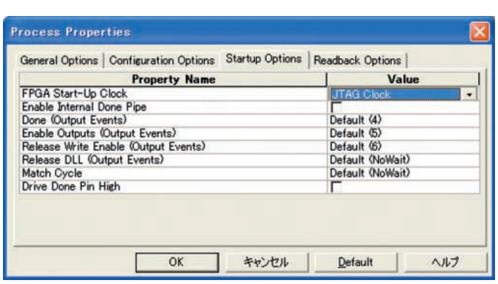


図23 Generate Programming Fileのプロパティ
FPGA基板のダウンロード方式に合わせて，「Startup Options」における「FPGA Start-up Clock」の「JATG Clock」を選択する．

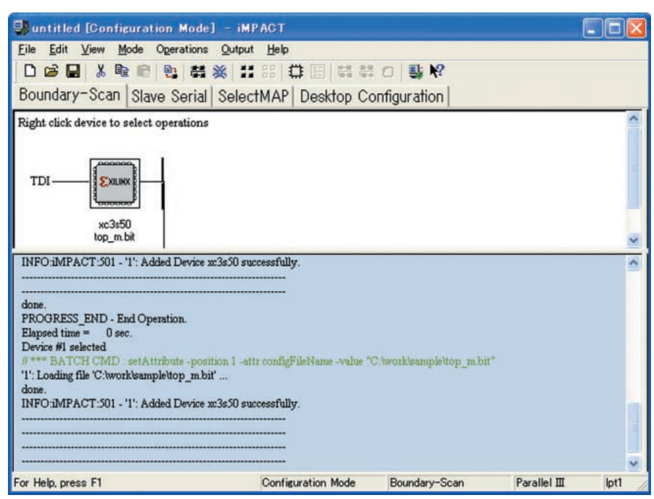


図24 iMPACTの画面例
メニューの下にXC3S50の絵が表示されなかったり，途中でエラーになる場合は，ダウンロード・ケーブルの接続を再確認する．

rate Programming File」の下の「Configure Device（iMPACT）」を選択し，マウスの右ボタンを押して，「Run」を選択して実行します．開いたダイアログの1ページ目では「Boundary-Scan Mode」を選択し，2ページ目では「Auotmatically ...」を選択します．iMPACTのウィンドウ（図24）が開きます．JTAGの認識に成功していれば，メニューの下にXC3S50の絵が表示されます．XC3S50をクリックして，「Edit」→「Assign Configuration File」を選択し，「top_m.bit」を指定します．次に，「Operations」→「Program」を選択して，［OK］ボタンを押すと，ダウンロードが始まります．途中でエラーになる場合は，ダウンロード・ケーブルの接続を再確認してください．

● 実機動作のチェックは外側から内側へ

シミュレーションの段階でしっかりと論理検証されていれば，一発で動作するはずです．しかし，実際にはなかなかそううまくいきません．ダウンロードが完了してもLCDモジュールに何も表示されない場合は，地道なチェックが必要です．実機による動作では，外側から内側に向かって確認していきます．

FPGA内部はシミュレーションで検証が済んでいることを前提として，まずはFPGAと外部の接続を確認します．ソース・コードには簡単なテスト回路が盛り込んであるので，それを活用します．初めにクロック入力を確認します．秒数のカウンタの1秒周期で変化する信号をLED2に接続してあります．ダウンロード後，LED2が1秒周期で点滅を繰り返せば，正常にクロック入力が行われています．点滅しない場合は，サブボードのクロック発生回路の周辺とピン割り当てを確認してください．

また，SW1～4の入力をANDして，LED1に接続してあるので，SW1～4のどれか一つを押すとLED1が点灯します．LED1の点灯によって，SW1～4とFPGAの接続を確認できます．点灯しないスイッチがある場合は，そのスイッチのサブボードの配線とFPGAのピン割り当てを確認します．

LCDモジュールが表示されない場合は，サブボードのLCD接続とFPGAのピン割り当てを確認してください．ここまで確認しても表示できない場合は，ソース・コードの見直しが必要です．最初に確認するのは，LCDモジュールを制御するLCD_CTRL.vです．LCDモジュールの初期化の手順や信号のタイミングを再確認してください．

スイッチからの入力とLCDモジュールの表示が正常で，外部に接続するブロックの確認が済んだら，内部ブロックの検証に移ります．SW4を押しながらリセットを解除すると，デバッグ・モードになります．LCDモジュールの右側に計算に使った演算を表示するので，動作確認時に活用してください．

うまく動かない部分があった場合には，ソース・コードを見直します．ソース・コードを変更した場合，めんどうでもシミュレーションで再度論理検証を行ってください．「実機はOKで，シミュレーションはNG」という状態になると，それ以降にソースを変更したとき，シミュレーションによる論理検証が不可能になり，困ったことになります．

● 自作ゲーム・マシンの作成にチャレンジ！

今回の設計では，演算当てゲーム専用の機能をMAINブロックに集めてあります．スイッチとLCDモジュールのインターフェースはそのままにして，MAINブロックの中身を変更すると，別の機能を実現できます．また，今回はRAMを使用しませんでしたが，XC3S50には72KビットのBlock RAMが搭載されています．RAMを使ったアプリケーションも作成可能です．

サブボードは数時間の作業で完成できる簡単なものです．自分専用のFPGA基板が手元にあると，設計でも何かと便利なことがあります．週末を利用して，自作ゲーム・マシンの作成にチャレンジしてみてください．

参考・引用*文献

(1) 小林優；入門Verilog-HDL記述，CQ出版社，1996年．
(2) 鳥海佳孝，田原迫仁治，横溝憲治；HDLサンプル記述集，CQ出版社，2002年．
(3) 編集部，近藤渉，西村憲二；総力特集 付属FPGA基板を使った回路設計チュートリアル Part2，第2章～第4章，Design Wave Magazine，2005年1月号，pp.29-53.

よこみぞ・けんじ
設計コンサルタント

<筆者プロフィール>
横溝憲治．1989年からNECエンジニアリングにてLSI設計および設計ワークステーションのシステム管理に従事．1996年から（有）ひまわりにて，HDL設計のトレーニング講師，受託設計，パソコン・ベースの中小企業向け業務システム構築などを請け負い，現在に至る．私生活では地域ボランティア活動に参加．最近はいろいろな意味でPTA活動にはまっている．エンジニアがPTA活動に参加できる道を模索中．